

# Applying FCA toolbox to Software Testing

Fedor Strok

Yandex

16 Leo Tolstoy St. Moscow, Russia

fdrstrok@yandex-team.ru

Faculty for Computer Science

National Research University Higher School of Economics

Kochnovskiy Proezd, 3, Moscow, Russia

fdr.strok@gmail.com

## ABSTRACT

Software testing uses wide range of different tools to enhance the complicated process of defining quality of the system under test. Formal Concept Analysis (FCA) provides us with algorithms of deriving formal ontology from a set of objects and their attributes. With the use of FCA we can considerably improve the efficiency of test case derivation. Moreover, an FCA-based machine learning system supports the analysis of regression testing results.

## 1. INTRODUCTION

Software testing aims at understanding the risks of software implementation and providing proper quality of the system under test. One of fundamental problems is that testing under all possible combinations of inputs is not feasible. This holds for functional testing, while non-functional aspects (performance, compatibility, etc.) are left apart. For business critical applications black-box testing is a widely-used approach. It examines the functionality of a system under test without dealing with its implementation. It is important to notice that static testing involves verification, while dynamic testing involves validation.

One of the biggest challenges in black-box dynamic testing is choosing the proper approach to enumerate all possible cases. Domain testing helps quality assurance engineer to define classes of input values, that are crucial to test. The naive way is just to check for possible combinations of such parameters, and it immediately leads to exponential complexity. Even in the case of 6 boolean parameters an expert has to check 64 combinations.

A possible solution is to generate test-cases in manual way. A software test engineer considers all possible cases ordered in a certain way. The main risk is to lose some information behind the cases. Actually, it could be rather exhausting to cover all possibilities for a large number of parameters and

not to skip some scenarios.

A popular alternative way is pairwise testing [2], or its generalization, n-wise testing. We define parameters and domains, and pass them like a model into a black-box algorithm [1], which gives us a set of test-cases satisfying a certain condition (for each pair of input parameters all possible discrete combinations of those parameters are tested). In general case it can produce different cases in different runs, while it can be fixed by passing a random seed to it. The main advantage of the approach is its insensitivity to parameter values. However, it can be a rather computationally demanding task.

For quality assurance engineers it is sufficient to have knowledge about possible behavior of the program. Usually, there are dependencies between input parameters. A natural form to express such dependencies in mathematical terms is implication, a statement of the form: ‘if ..., then ...’. The ‘if’-part is called premise, and the ‘then’ is called conclusion. Consideration of parameter’s interdependence can decrease the complexity of result test-cases in terms of their quantity, by excluding some of possible parameter combinations.

Proposed approach to test-case generation is focused on implications. We use an approach based on Formal Concept Analysis (FCA)[5]. FCA provides software engineers with a tool for exploring the domain of interest in semi-automatic way. The algorithm outputs sound and complete description of the problem, but it is still highly dependent on the accuracy of expert’s answers. Surveys on FCA techniques and applications can be found in [11]. FCA-based approaches have already been applied in software engineering, e.g., for inference of class hierarchies [12], class design [13], refactoring [14]. Another application of FCA technique is analysis of object-oriented approach for a given system [15]. The questions of mapping lines of source code to the functionality from requirements is of crucial importance for big systems [16].

The rest of the paper is organized as follows: in Section 2 we introduce basic notions of Formal Concept Analysis. In Section 3 we focus on the procedure of attribute exploration. Section 4 provides example of attribute exploration in the field of positive integers. We make conclusions in Section 5.

## 2. FORMAL CONCEPT ANALYSIS





Formal Concept Analysis [5] (FCA) provides mathematical technique for deriving applied ontologies from data. FCA relies on lattice and order theories [3] Numerous applications are found in the field of machine learning, data mining, text mining and biology, see [11, ?].

## 2.1 Main Definitions

*Definition 1.* A formal context  $K$  is a triple  $K := (G, M, I)$ , where  $G$  denotes a set of objects,  $M$  is a set of attributes, and  $I \subseteq G \times M$  is a binary relation between  $G$  and  $M$ .

It can be interpreted in the following way: for objects in  $G$  there exists a description in terms of attributes in  $M$ , and relation  $I$  reflects that an object has an attribute:  $(g, m) \in I \iff \text{object } g \text{ possesses } m$ .

An example of a formal context is provided below:

| G \ M   | a | b | c | d |
|---|---|---|---|---|
|  | x |   |   | x |
|  | x |   | x |   |
|  |   | x | x |   |
|  |   | x | x | x |

Objects:

- 1 – equilateral triangle
- 2 – right triangle,
- 3 – rectangle,
- 4 – square,

Attributes:

- a – 3 vertices,
- b – 4 vertices,
- c – has a right angle,
- d – all sides are equal

For a given context two following mappings are considered:

$$\varphi: 2^G \rightarrow 2^M \quad \varphi(A) \stackrel{\text{def}}{=} \{m \in M \mid gIm \text{ for all } g \in A\}.$$

$$\psi: 2^M \rightarrow 2^G \quad \psi(B) \stackrel{\text{def}}{=} \{g \in G \mid gIm \text{ for all } m \in B\}.$$

For all  $A_1, A_2 \subseteq G$ ,  $B_1, B_2 \subseteq M$

1.  $A_1 \subseteq A_2 \Rightarrow \varphi(A_2) \subseteq \varphi(A_1)$
2.  $B_1 \subseteq B_2 \Rightarrow \psi(B_2) \subseteq \psi(B_1)$
3.  $A_1 \subseteq \psi\varphi(A_1)$  and  $B_1 \subseteq \varphi\psi(B_1)$

*Definition 2.* Mappings  $\varphi$  and  $\psi$ , satisfying properties 1-3 above, define a Galois connection between  $(2^G, \subseteq)$  and  $(2^M, \subseteq)$ , which means:  $\varphi(A) \subseteq B \Leftrightarrow \psi(B) \subseteq A$

Traditionally, notation  $(\cdot)'$  is used instead of  $\varphi$  and  $\psi$ .  $(\cdot)''$  stands both for  $\varphi \circ \psi$  and  $\psi \circ \varphi$  (depending on the argument). For arbitrary  $A \subseteq G$ ,  $B \subseteq M$

$$A' \stackrel{\text{def}}{=} \{m \in M \mid gIm \text{ for all } g \in A\},$$

$$B' \stackrel{\text{def}}{=} \{g \in G \mid gIm \text{ for all } m \in B\}.$$

*Definition 3.* (Formal) concept is a pair  $(A, B)$ :  $A \subseteq G$ ,  $B \subseteq M$ ,  $A' = B$ ,  $B' = A$ .

In the example with geometric figures a pair  $(\{3, 4\}, \{b, c\})$  is a formal concept. For a formal context  $(G, M, I)$ ,  $A, A_1, A_2 \subseteq G$  - set of objects,  $B \subseteq M$  - set of attributes, the following statements hold for operation  $(\cdot)'$ :

1.  $A_1 \subseteq A_2 \Rightarrow A_2' \subseteq A_1'$
2.  $A_1 \subseteq A_2 \Rightarrow A_1'' \subseteq A_2''$
3.  $A \subseteq A''$
4.  $A''' = A'$  and  $A'''' = A''$
5.  $(A_1 \cup A_2)' = A_1' \cap A_2'$
6.  $A \subseteq B' \Leftrightarrow B \subseteq A' \Leftrightarrow A \times B \subseteq I$

*Definition 4.* Closure operator on set  $G$  is a mapping  $\gamma: \mathcal{P}(G) \rightarrow \mathcal{P}(G)$ , which maps every  $X \subseteq G$  to  $\text{closure } \gamma X \subseteq G$ , under the following conditions:

1.  $\gamma\gamma X = \gamma X$  (*idempotence*)
2.  $X \subseteq \gamma X$  (*extensivity*)
3.  $X \subseteq Y \Rightarrow \gamma X \subseteq \gamma Y$  (*monotonicity*)

*Definition 5.* Implication  $A \rightarrow B$ , where  $A, B \subseteq M$ , takes place if  $A' \subseteq B'$ , in other words if each object having  $A$  also has all attributes from  $B$ .

Implications comply with Armstrong axioms:

$$\overline{X \rightarrow X}^{(1)}$$

$$\frac{X \rightarrow Y}{X \cup Z \rightarrow Y}^{(2)}$$

$$\frac{X \rightarrow Y, Y \cup Z \rightarrow W}{X \cup Z \rightarrow W}^{(3)}$$

## 3. LATTICE DIAGRAMS

One of advantages of building lattices of formal concepts is to get effective navigation from more general concepts to more specific. For example, the line diagram for context with figures from previous section is shown in Fig. 1. That property could be beneficial in two ways: regression testing, system description.

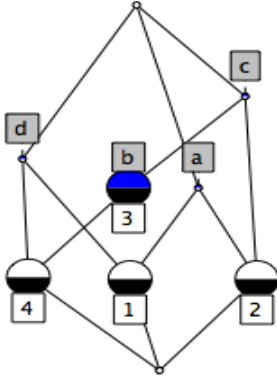


Figure 1: Line diagram for context with geometric figures

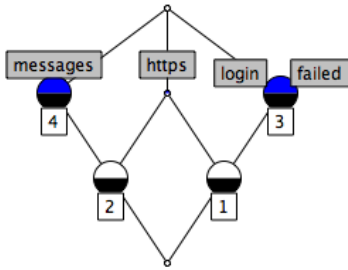


Figure 2: Test run: Line diagram of all tests

First of all, for the context of regression testing it implements the algorithm to determine the classes of tests that fail. It could be considered in the following way. Let us define set of attributes for all tests that are being run.

| G \ M | failed | https | login | messages |
|-------|--------|-------|-------|----------|
| 1     | ×      | ×     | ×     |          |
| 2     |        | ×     |       | ×        |
| 3     | ×      |       | ×     |          |
| 4     |        |       |       | ×        |

Then we can view the line diagram of all the tests during the run. See Fig. 2. The most interesting formal concepts are those with attribute “failed=True”, since the simplified result could be treated as either success or fail. Now we can notice that “messages” part is ok, while login module breaks totally, and there is one problem with “https”. The convenient strategy for making such meta-reports is to filter the set of all tests which have “failed=True” flag. Then we

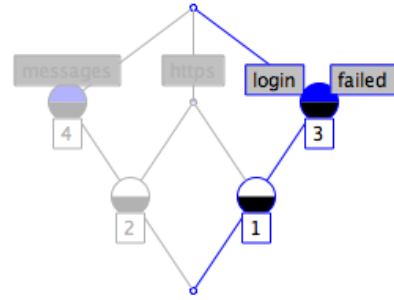


Figure 3: Test run: line diagram of Concepts with failed=True highlighted

can build concept lattice for obtained filtered formal context. Moreover, we could build just the top part of concept lattice, since it will have most general attributes.

The second important application of formal concepts in big systems is analysis of dependent features. Since modern software development process usually is organized in agile manner, it is important to view the effect of developed features on the overall system, and examine connected components. It could be fruitful to use test cases from features with common functionality. To track such features and to have user-friendly navigation we can use FCA-based techniques. We need to have the list of features and set of tags as general components description. A small example is provided below. We can build formal concept lattice for the context describing the features. Then we can search obtained graph, finding not only features with the same set of tags, but also more general and more specific ones. The initial lattice is depicted in Fig. 4. If we want to analyze the feature with tags “https” and “login” we see that it forms formal concept  $(\{f1, f3, f5\}, \{ \})$

| G \ M | billing | https | login | messages | static |
|-------|---------|-------|-------|----------|--------|
| f1    | ×       | ×     | ×     |          |        |
| f2    |         | ×     |       | ×        |        |
| f3    | ×       | ×     | ×     |          | ×      |
| f4    |         |       | ×     | ×        |        |
| f5    |         | ×     | ×     |          | ×      |

#### 4. FEATURE IMPACT ANALYSIS

One of the most important steps of testing software is to determine the functionality that would be changed. Usually, all activities concerning the development process are stored in issue trackers. It is important to reuse the experience of previous features while developing new functionality. We can think of two dimensions of analysis of related features. Firstly, issue trackers typically provide the possibility to link tickets and in this way we can draw graph of similar features. Secondly, we can benefit from tags that are used to mark system parts. So, example from the previous section could be studied in alternative way: we can browse features to find

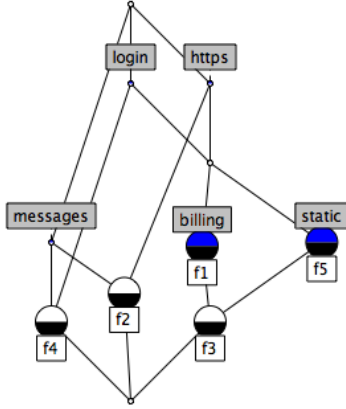


Figure 4: Features description

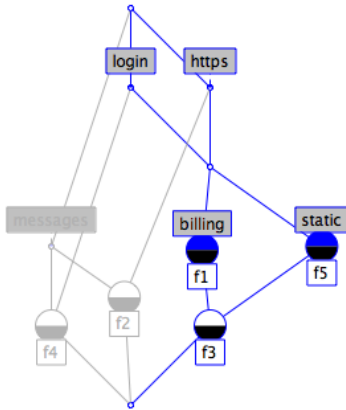


Figure 5: Features description: concept with “https” and “login”

the closest one in terms of functional units. We can find out which cases were obligatory in this case and simplify test-case design process.

## 5. ATTRIBUTE EXPLORATION

Attribute exploration is a well-known FCA-based discovery technique. The main idea is to explore the implications in the object domain in a semi-automatic manner. According to Attribute Exploration a domain expert answers specific questions about possible implicational dependencies in his domain. Questions are provided in the form of attribute implications, asking whether they are true or false. If the answer is true, the implication is added to the knowledge base. In the case of answer “false”, the expert is asked to provide a counterexample violating the proposed dependency.

In other words, the exploration algorithm explores all possible combinations of a given attribute set. It is typical that objects in this field of knowledge are too difficult to enumerate them. So the algorithm starts with a set of examples, then it computes canonical base of implications for the provided formal context. Then the domain expert is asked if the computed implications are valid. If it is true, then the existing context represents all possible combinations in the domain. Otherwise, the expert has to provide a counterexample to implications, which should be added to the context as new objects, and then the implication base is recomputed.

The general strategy is quite intuitive: we start exploring the domain with some knowledge of typical examples and implications. To extend the knowledge base we either add an implication, or provide another example that violates currently studied implication. The main advantage of this approach is that generation of dependencies (implications) is performed algorithmically.

---

**Algorithm 1** NEXT CLOSURE( $A, M, \mathcal{L}$ )

---

**Input:** Closure operator  $X \mapsto \mathcal{L}(X)$  on attribute set  $M$  and subset  $A \subseteq M$ .

**Output:** lexicographically next closed itemset  $A$ .

for all  $m \in M$  in reverse order do

if  $m \in A$  then

$A := A \setminus \{m\}$

else

$B := \mathcal{L}(A \cup \{m\})$

if  $B \setminus A$  does not contain elements  $< m$  then

return  $B$

return  $\perp$

---

---

**Algorithm 2** ATTRIBUTE EXPLORATION

---

**Input:** A subcontext  $(E, M, J = I \cap E \times M)$  of  $(G, M, I)$ , possibly empty.

**Input:** Interactive: confirm that  $A = B''$  in a formal context  $(G, M, I)$ ,  $M$  finite, or give an object showing that  $A \neq B''$ .

**Output:** The canonical base  $\mathcal{L}$  of  $(G, M, I)$  and a possibly enlarged subcontext  $(E, M, J = I \cap E \times M)$  with the same canonical base.

$\mathcal{L} := \emptyset$

$A := \emptyset$

**while**  $A \neq M$  **do**

**while**  $A \neq A^{JJ}$  **do**

**if**  $A^{JJ} = A^{II}$  **then**

$\mathcal{L} := \mathcal{L} \cup \{A \rightarrow A^{JJ}\}$

**exit while**

**else**

        extend  $E$  by some object  $g \in A^I \setminus A^{JJ}$

$A := \text{NextClosure}(A, M, \mathcal{L})$

**return**  $\mathcal{L}, (E, M, J)$

---

## 6. PRACTICAL EXAMPLES

### 6.1 Numbers

Let us consider the domain of natural numbers [6]. As a set of possible attributes we can consider the following ones: even ( $2*n$ ), odd ( $2*n+1$ ), divisible\_by\_three ( $3*n$ ), prime (has no positive divisors other than 1 and itself), factorial (is a factorial of a positive number). We start from the empty set of objects. The canonical base for such context is  $\emptyset \rightarrow M$ . So we get a question:

$\Rightarrow \text{even}, \text{factorial}, \text{divided\_by\_three}, \text{odd}, \text{prime}$

Is the following implication valid?

Obviously, not all numbers have all attributes. At least we can consider number 2, which is *even, factorial, prime*. We add 2 to our context and the base is recomputed.

| G \ M | even | factorial | divided_by_three | odd | prime |
|-------|------|-----------|------------------|-----|-------|
| 2     | ×    | ×         |                  |     | ×     |

$\Rightarrow \text{even}, \text{factorial}, \text{prime}$

Is the following implication valid?

Now we can consider number 5, which is *prime, odd*

| G \ M | even | factorial | divided_by_three | odd | prime |
|-------|------|-----------|------------------|-----|-------|
| 2     | ×    | ×         |                  |     | ×     |
| 5     |      |           |                  | ×   | ×     |

$\Rightarrow \text{prime}$

Is the following implication valid?

Now we are about to either say that all numbers are prime, or provide a non-prime number, e.g. 6

| G \ M | even | factorial | divided_by_three | odd | prime |
|-------|------|-----------|------------------|-----|-------|
| 2     | ×    | ×         |                  |     | ×     |
| 5     |      |           |                  | ×   | ×     |
| 6     | ×    | ×         | ×                |     |       |

$\text{factorial} \Rightarrow \text{even}$

Is the following implication valid?

Now we have both 2 and 6, which are simultaneously even and factorial. There is a counterexample, we should find a number that is factorial, but not even, which is 1.

| G \ M | even | factorial | divided_by_three | odd | prime |
|-------|------|-----------|------------------|-----|-------|
| 2     | ×    | ×         |                  |     | ×     |
| 5     |      |           |                  | ×   | ×     |
| 6     | ×    | ×         | ×                |     |       |
| 1     |      | ×         |                  | ×   | ×     |

$\text{odd} \Rightarrow \text{prime}$

Is the following implication valid?

No, it does not hold for number 9.

| G \ M | even | factorial | divided_by_three | odd | prime |
|-------|------|-----------|------------------|-----|-------|
| 2     | ×    | ×         |                  |     | ×     |
| 5     |      |           |                  | ×   | ×     |
| 6     | ×    | ×         | ×                |     |       |
| 1     |      | ×         |                  | ×   | ×     |
| 9     |      |           | ×                | ×   |       |

$\text{factorial}, \text{odd} \Rightarrow \text{prime}$

Is the following implication valid?

We have the only number 1 which is factorial and odd, and it is also prime.

$\text{factorial}, \text{divided\_by\_three} \Rightarrow \text{even}$

Is the following implication valid?

Now we have to recall what is implication? The only case where implication does not hold is when premise is true and conclusion is false. The least factorial which is divided\_by\_three is 6, which is already even.

$\text{prime}, \text{divided\_by\_three} \Rightarrow \text{even}, \text{factorial}, \text{odd}$

Is the following implication valid?

We have number 3, which is just odd.

| G \ M | even | factorial | divided_by_three | odd | prime |
|-------|------|-----------|------------------|-----|-------|
| 2     | ×    | ×         |                  |     | ×     |
| 5     |      |           |                  | ×   | ×     |
| 6     | ×    | ×         | ×                |     |       |
| 1     |      | ×         |                  | ×   | ×     |
| 9     |      |           | ×                | ×   |       |
| 3     |      |           | ×                | ×   | ×     |

$\text{prime}, \text{divided\_by\_three} \Rightarrow \text{odd}$

Is the following implication valid?

The only prime that is divided\_by\_three is three itself, so it is true.  $\text{even} \Rightarrow \text{factorial}$

Is the following implication valid?

Not all even numbers are factorials, e.g. 8.

| G \ M | even | factorial | divided_by_three | odd | prime |
|-------|------|-----------|------------------|-----|-------|
| 2     | ×    | ×         |                  |     | ×     |
| 5     |      |           |                  | ×   | ×     |
| 6     | ×    | ×         | ×                |     |       |
| 1     |      | ×         |                  | ×   | ×     |
| 9     |      |           | ×                | ×   |       |
| 3     |      |           | ×                | ×   | ×     |
| 8     | ×    |           |                  |     |       |

$even, odd \Rightarrow factorial, prime, divided\_by\_three$

Is the following implication valid?

We do not have numbers which are both even and odd.

$even, divided\_by\_three \Rightarrow factorial$

Is the following implication valid?

We have number 12, which is even and divided\_by\_three, but it is not a factorial.

| G \ M | even | factorial | divided_by_three | odd | prime |
|-------|------|-----------|------------------|-----|-------|
| 2     | ×    | ×         |                  |     | ×     |
| 5     |      |           |                  | ×   | ×     |
| 6     | ×    | ×         | ×                |     |       |
| 1     |      | ×         |                  | ×   | ×     |
| 9     |      |           | ×                | ×   |       |
| 3     |      |           | ×                | ×   | ×     |
| 8     | ×    |           |                  |     |       |
| 12    | ×    |           | ×                |     |       |

$even, prime \Rightarrow factorial$

Is the following implication valid?

The only even prime number is 2. So, the exploration process is over.

The final context:

| G \ M | even | factorial | divided_by_three | odd | prime |
|-------|------|-----------|------------------|-----|-------|
| 1     |      | ×         |                  | ×   | ×     |
| 2     | ×    | ×         |                  |     | ×     |
| 3     |      |           | ×                | ×   | ×     |
| 5     |      |           |                  | ×   | ×     |
| 6     | ×    | ×         | ×                |     |       |
| 8     | ×    |           |                  |     |       |
| 9     |      |           | ×                | ×   |       |
| 12    | ×    |           | ×                |     |       |

The set of implications:

- $factorial, odd \rightarrow prime$
- $factorial, divided\_by\_three \rightarrow even$
- $prime, divided\_by\_three \rightarrow odd$
- $even, odd \rightarrow factorial, prime, divided\_by\_three$
- $even, prime \rightarrow factorial$

## 6.2 Numbers: model-based

We can consider the same problem and use pairwise testing. One of known tools for this is PICT, pairwise testing tool by Microsoft. The initial model is quite simple.

- Event: 1, 0
- Factorial: 1, 0
- Divs3: 1, 0
- Odd: 1, 0
- Prime: 1, 0

The result of pairwise generation is shown in table below:

| Even | Factorial | Divs3 | Odd | Prime |
|------|-----------|-------|-----|-------|
| 1    | 1         | 0     | 0   | 1     |
| 0    | 0         | 1     | 1   | 0     |
| 1    | 0         | 1     | 1   | 1     |
| 0    | 1         | 0     | 1   | 0     |
| 1    | 0         | 1     | 0   | 0     |
| 0    | 1         | 1     | 0   | 1     |
| 0    | 0         | 0     | 0   | 0     |

To get the results as in the previous section, we have to modify the input model in the following way:

### 1. Parameters:

- Event: 1, 0
- Factorial: 1, 0
- Divs3: 1, 0
- Odd: 1, 0
- Prime: 1, 0
- Result: 12, 9, 6, 3, 2, 1

### 2. Implications

- IF [Even] = 1 THEN [Odd] = 0 ELSE [Odd] = 1;
- IF [Odd] = 1 AND [Factorial] = 1 THEN [Result] = 1;
- IF [Even] = 1 AND [Prime] = 1 THEN [Result] = 2;
- IF [Divs3] = 1 AND [Prime] = 1 THEN [Result] = 3;
- IF [Divs3] = 1 AND [Even] = 1 THEN [Result] IN 6, 12;
- IF [Divs3] = 1 AND [Odd] = 1 AND [Prime] = 0 THEN [Result] = 9;
- IF [Even] = 1 AND [Factorial] = 1 AND [Divs3] = 0 THEN [Result] = 2;

### 3. Data-specific dependencies

- IF [Result] = 1 THEN [Even] = 0 AND [Factorial] = 1 AND [Divs3] = 0 AND [Odd] = 1 AND [Prime] = 1;

- IF [Result] = 2 THEN [Even] = 1 AND [Factorial] = 1 AND [Divs3] = 0 AND [Odd] = 0 AND [Prime] = 1;
- IF [Result] = 3 THEN [Even] = 0 AND [Factorial] = 0 AND [Divs3] = 1 AND [Odd] = 1 AND [Prime] = 1;
- IF [Result] = 6 THEN [Even] = 1 AND [Factorial] = 1 AND [Divs3] = 1 AND [Odd] = 0 AND [Prime] = 0;
- IF [Result] = 9 THEN [Even] = 0 AND [Factorial] = 0 AND [Divs3] = 1 AND [Odd] = 1 AND [Prime] = 0;
- IF [Result] = 12 THEN [Even] = 1 AND [Factorial] = 0 AND [Divs3] = 1 AND [Odd] = 0 AND [Prime] = 0;

For such model PICT outputs the following cases:

| Even | Factorial | Divs3 | Odd | Prime | Result |
|------|-----------|-------|-----|-------|--------|
| 0    | 0         | 1     | 1   | 1     | 3      |
| 1    | 1         | 1     | 0   | 0     | 6      |
| 0    | 0         | 1     | 1   | 0     | 9      |
| 1    | 0         | 1     | 0   | 0     | 12     |
| 1    | 1         | 0     | 0   | 1     | 2      |
| 0    | 1         | 0     | 1   | 1     | 1      |

### 6.3 Banners with geotargeting

One of the most popular forms of advertising in the Internet is contextual advertisement. The key point is that companies pay for the click on their sites, but not for the shows, so the targeting part gains more importance. One of most-widely used features is geotargeting, i.e., defining in which regions the banner could be shown. Usually, regions are treated like a tree and if banner targets at some region, then it could be shown in it, and all it subregions. The possible sources for defining region: query argument, search query, ip address. Attributes with binary domains:  $M = \{IsQueryPresent(IQP), IsQueryValid(IQV), IsSqPresent(ISP), IsSqValid(ISV), IsIpPresent(IIP), IsIpValid(IIV), UserRegion == BannerRegion(=), UserRegion < BannerRegion(<), UserRegion > BannerRegion(>), IsBannerShown(IBS)\}$

|    | IQP | IQV | ISP | ISV | IIP | IIV | = | < | > | IBS |
|----|-----|-----|-----|-----|-----|-----|---|---|---|-----|
| 1  | ×   |     |     |     |     |     |   |   |   |     |
| 2  |     |     | ×   |     |     |     |   |   |   |     |
| 3  |     |     |     |     | ×   |     |   |   |   |     |
| 4  | ×   | ×   |     |     |     |     | × |   |   | ×   |
| 5  | ×   | ×   |     |     |     |     |   | × |   | ×   |
| 6  |     |     | ×   | ×   |     |     | × |   |   | ×   |
| 7  |     |     | ×   | ×   |     |     |   | × |   | ×   |
| 8  | ×   | ×   |     |     |     |     |   |   | × |     |
| 9  |     |     | ×   | ×   |     |     |   |   | × |     |
| 10 |     |     |     |     | ×   | ×   | × |   |   | ×   |
| 11 |     |     |     |     | ×   | ×   |   | × |   | ×   |
| 12 |     |     |     |     | ×   | ×   |   |   | × |     |
| 13 | ×   | ×   | ×   | ×   |     |     | × |   |   | ×   |
| 14 | ×   | ×   | ×   | ×   |     |     |   | × |   | ×   |
| 15 | ×   | ×   | ×   | ×   |     |     |   |   | × |     |
| 16 | ×   | ×   |     |     | ×   | ×   | × |   |   | ×   |
| 17 | ×   | ×   |     |     | ×   | ×   |   | × |   | ×   |
| 18 | ×   | ×   |     |     | ×   | ×   |   |   | × |     |
| 19 |     |     | ×   | ×   | ×   | ×   | × |   |   | ×   |
| 20 |     |     | ×   | ×   | ×   | ×   |   | × |   | ×   |
| 21 |     |     | ×   | ×   | ×   | ×   |   |   | × |     |
| 22 | ×   |     | ×   |     |     |     |   |   |   |     |
| 23 | ×   |     |     |     | ×   |     |   |   |   |     |
| 24 |     |     | ×   |     | ×   |     |   |   |   |     |
| 25 | ×   |     | ×   |     | ×   |     |   |   |   |     |
| 26 | ×   |     | ×   |     | ×   | ×   | × |   |   | ×   |
| 27 | ×   | ×   | ×   |     |     |     | × |   |   | ×   |
| 28 | ×   | ×   | ×   |     | ×   |     | × |   |   | ×   |
| 29 | ×   | ×   | ×   |     | ×   | ×   | × |   |   | ×   |
| 30 | ×   |     | ×   | ×   |     |     | × |   |   | ×   |
| 31 | ×   |     | ×   | ×   | ×   |     | × |   |   | ×   |
| 32 | ×   |     | ×   | ×   | ×   | ×   | × |   |   | ×   |
| 33 | ×   | ×   | ×   | ×   | ×   | ×   | × |   |   | ×   |
| 34 | ×   | ×   |     |     | ×   | ×   |   | × |   | ×   |
| 35 | ×   | ×   |     |     | ×   | ×   |   |   | × |     |
| 36 | ×   |     | ×   | ×   |     |     |   | × |   | ×   |
| 37 | ×   |     | ×   | ×   |     |     |   |   | × |     |
| 38 | ×   |     | ×   |     | ×   | ×   |   | × |   | ×   |
| 39 | ×   |     | ×   |     | ×   | ×   |   |   | × |     |
| 40 | ×   |     |     |     | ×   | ×   |   | × |   | ×   |
| 41 | ×   |     |     |     | ×   | ×   |   |   | × |     |
| 42 |     |     | ×   |     | ×   | ×   | × |   |   | ×   |
| 43 |     |     | ×   | ×   | ×   |     | × |   |   | ×   |
| 44 | ×   | ×   | ×   |     | ×   |     |   | × |   | ×   |
| 45 | ×   | ×   | ×   |     | ×   |     |   |   | × |     |
| 46 | ×   |     | ×   | ×   | ×   |     |   | × |   | ×   |
| 47 | ×   |     | ×   | ×   | ×   |     |   |   | × |     |
| 48 | ×   | ×   | ×   | ×   | ×   |     | × |   |   | ×   |
| 49 | ×   | ×   | ×   | ×   | ×   |     |   | × |   | ×   |
| 50 | ×   | ×   | ×   | ×   | ×   |     |   |   | × |     |
| 51 | ×   | ×   | ×   |     | ×   | ×   |   | × |   | ×   |
| 52 | ×   | ×   | ×   |     | ×   | ×   |   |   | × |     |
| 53 | ×   |     | ×   | ×   | ×   | ×   |   | × |   | ×   |
| 54 | ×   |     | ×   | ×   | ×   | ×   |   |   | × |     |
| 55 | ×   | ×   | ×   | ×   | ×   | ×   |   | × |   | ×   |
| 56 | ×   | ×   | ×   | ×   | ×   | ×   |   |   | × |     |
| 57 |     |     | ×   |     | ×   | ×   |   | × |   | ×   |
| 58 |     |     | ×   |     | ×   | ×   |   |   | × |     |
| 59 | ×   | ×   |     |     | ×   |     | × |   |   | ×   |
| 60 | ×   |     |     |     | ×   | ×   | × |   |   | ×   |
| 61 |     |     |     |     |     |     |   |   |   |     |
| 62 |     |     | ×   | ×   | ×   |     |   | × |   | ×   |
| 63 |     |     | ×   | ×   | ×   |     |   |   | × |     |
| 64 | ×   | ×   |     |     | ×   |     |   | × |   | ×   |
| 65 | ×   | ×   |     |     | ×   |     |   |   | × |     |
| 66 | ×   | ×   | ×   |     |     |     |   | × |   | ×   |
| 67 | ×   | ×   | ×   |     |     |     |   |   | × |     |

Implication list:

1. **IF** IQV **THEN** IQP
2. **IF** ISV **THEN** ISP
3. **IF** IIV **THEN** IIP
4. **IF** = **THEN** IBS, not <, not >
5. **IF** < **THEN** IBS, not =, not >
6. **IF** IBS **THEN** not >
7. **IF** not IQP **THEN** not IQV
8. **IF** not ISP **THEN** not ISV
9. **IF** not IIP **THEN** not IIV
10. **IF** not IIV, not ISV, not IQV **THEN** not =, not <, not >, not IBS
11. **IF** not <, not = **THEN** not IBS
12. **IF** not >, IQV, IQP **THEN** IBS
13. **IF** not >, ISV, ISP **THEN** IBS
14. **IF** not >, IIV, IIP **THEN** IBS
15. **IF** not >, not ISV, not IQV, IBS **THEN** IIV
16. **IF** not >, not ISV, not IIV, IBS **THEN** IQV
17. **IF** not >, not IIV, not IQV, IBS **THEN** ISV
18. **IF** not >, not =, IBS **THEN** <
19. **IF** not >, not <, IBS **THEN** =
20. **IF** not IBS **THEN** not =, not <
21. **IF** not IBS, not <, not =, IQV, IQP **THEN** >
22. **IF** not IBS, not <, not =, ISV, ISP **THEN** >
23. **IF** not IBS, not <, not =, IIV, IIP **THEN** >
24. **IF** not IBS, not <, not =, >, not IQV, not ISV **THEN** IIV, IIP
25. **IF** not IBS, not <, not =, >, not IIV, not IQV **THEN** ISV, ISP
26. **IF** not IBS, not <, not =, > not IIV, not ISV **THEN** IQV, IQP
27. **IF** not IBS, not >, not =, not < **THEN** not IQV, not ISV, not IIV

## 7. PLUG-IN SETUP

The most important advantage of the proposed approach is the plug-in design. It can be easily incorporated in existing process of test-case generation. For manual test-case design one can develop test-cases in proposed system. Moreover, the step of implication extraction could be postponed up to the review of obtained cases. It is important to notice that Attribute Exploration is a good technique to extract dependencies that could be formulated as requirements for the system under test. For the case of automatic test case generation, extraction of dependencies could eliminate the step of test debug and replace it with review of obtained implications and the model of the system under test. Also the proposed technique is applicable to verify the correctness of defining type of model, precisely wiseness of it. Since big values of  $n$  in  $n$ -wise modeling impose more test cases and it results in the growing time of test run execution.

## 8. CONCLUSION

Formal Concept Analysis provides us with useful framework for software testing tasks. It is especially beneficial in regression testing meta report construction, feature navigation, test case analysis and derivation. It unites best practices of manual development and automatic generation of test scenarios. It provides sound and complete description of the investigated domain, based on expert knowledge. The output of the system consists of two main parts: the description of typical objects in the domain, and interdependence between parameters in terms of implications. An important advantage of proposed technique is extensibility. If we add a new attribute, we can just copy the previous examples into new formal context, assuming that new attribute is absent for all objects and proceed with the procedure of attribute exploration. It holds even for the very start of procedure. We can start with non-empty set of objects and implications simultaneously. The described algorithm could be used as a standalone solution for the test case design, as well as, tool to get exiting dependencies in the domain. The obtained implications could be valuable in pairwise testing to adjust the model description.

However, we should admit that the current approach is limited in terms of attribute description. For now, it is highly dependent on the boolean nature of attributes. One of the main directions of future work is to work with descriptions of general form by means of patterns structures [4, 10], an extension of FCA.

## 9. REFERENCES

- [1] Bach, J. and Shroeder, P. 2004. Pairwise Testing - A Best Practice That Isn't. In Proceedings of the 22nd Pacific Northwest Software Quality Conference, 2004.
- [2] Czerwinka, J. 2006. Pairwise Testing in Real World: Practical Extensions to Test Case Generators. In Proceedings of the 24th Pacific Northwest Software Quality Conference
- [3] Davey, B. and Priestly, H. *Introduction to Lattices and Order (2nd edition)*. Cambridge Mathematical Textbooks. Cambridge University Press, 2002.
- [4] Ganter, B. and Kuznetsov, S.O., Pattern Structures and Their Projections. In: Proc. G. Stumme and H. Delugach, Eds., 9th International Conference on



Conceptual Structures (ICCS 2001), Lecture Notes in Artificial Intelligence (Springer), Vol. 2120, pp. 129-142, 2001.

- [5] Ganter, B. and Wille, R., *Formal Concept Analysis: Mathematical Foundations*, Springer, 1999.
- [6] <https://github.com/ae-hse/fca/>
- [7] Revenko, A. and Kuznetsov, S. O. (2012), Finding Errors in New Object Intents., in Laszlo Szathmary and Uta Priss, ed., 'CLA' , CEUR-WS.org, , pp. 151-162 .
- [8] Bazhanov K. and Obiedkov S. Optimizations in computing the Duquenne-Guigues basis of implications // Annals of Mathematics and Artificial Intelligence. 2014. Vol. 70. No. 1-2. P. 5-24.
- [9] Kuznetsov S. O., Revenko A. Interactive Error Correction in Implicative Theories //arXiv preprint arXiv:1410.5215. - 2014.
- [10] Kuznetsov, S.O., Fitting Pattern Structures for Knowledge Discovery in Big Data. In: Cellier, Peggy; Distel, Felix; Ganter, Bernhard, Eds., Proc. 11th International Conference on Formal Concept Analysis (ICFCA 2013), Lecture Notes in Artificial Intelligence (Springer), Vol. 7880, pp. 254 – 266, 2013.
- [11] Poelmans J., Kuznetsov S. O., Ignatov D. I., Dedenea G., Formal Concept Analysis in knowledge processing: A survey on models and techniques. In: Expert Systems with Applications, Vol. 40. No. 16, pp. 6601-6623, 2013.
- [12] Snelting, G., Tip, F. (2000). Understanding class hierarchies using concept analysis. ACM Transactions on Programming Languages and Systems (TOPLAS), 22(3), 540-582.
- [13] Godin, R., Valtchev, P. (2004). Formal concept analysis-based normal forms for class hierarchy design in oo software development. Formal Concept Analysis: Foundations and Applications, 304-323.
- [14] Moha, N., Hacene, A. M. R., Valtchev, P., Gueheneuc, Y. G. (2008). Refactorings of design defects using relational concept analysis. In Formal Concept Analysis (pp. 289-304). Springer Berlin Heidelberg.
- [15] Tilley, T., Cole, R., Becker, P., Eklund, P. (2005). A survey of formal concept analysis support for software engineering activities. In Formal concept analysis (pp. 250-271). Springer Berlin Heidelberg.
- [16] Poshyvanyk, D., Marcus, A. (2007, June). Combining formal concept analysis with information retrieval for concept location in source code. In Program Comprehension, 2007. ICPC'07. 15th IEEE International Conference on (pp. 37-48). IEEE.